COPY

(2)

hen Data Entered)

AD-A220 348

TATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) **Ada Compiler Validation Summary Report:** R.R. Software, Inc., Janus/ADA, Version 2.1.3, Unisys 386 (Host & Target), 890919W1.10153	5. TYPE OF REPORT & PERIOD COVERED 19 Sept. 1989 to 19 Sept. 1990 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson AFB Dayton, OH, USA	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson AFB Dayton, OH, USA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Wright-Patterson AFB Dayton, OH, USA	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

R. R. Software, Inc., Janus/ADA, Version 2.1.3, Wright-Patterson AFB, Unisys 386 under SCO Xenix, Version 2.3.2 (Host & Target), ACVC 1.10

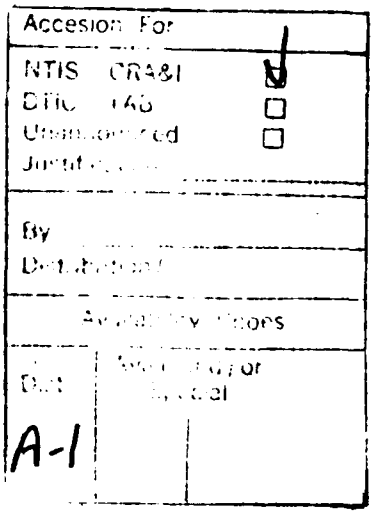
AVF Control Number: AVF-VSR-317.0290
89-07-18-RRS

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 890919W1.10153
R.R. Software, Inc.
Janus/ADA, Version 2.1.3
Unisys 386

Completion of On-Site Testing:
19 September 1989

Prepared By:
Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington DC 20301-3081



Ada Compiler Validation Summary Report:

Compiler Name: Janus/ADA, Version 2.1.3

Certificate Number: 890919W1.10153

Host: Unisys 386 under
SCO Xenix, Version 2.3.2

Target: Unisys 386 under
SCO Xenix, Version 2.3.2

Testing Completed 19 September 1989 Using ACVC 1.10

Customer Agreement Number: 89-07-18-RRS

This report has been reviewed and is approved.

Ada Validation Facility
Steven P. Wilson
Technical Director
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

for Ada Validation Organization
Director, Computer & Software Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 19 September 1989 at Madison WI.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C.#552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SCOL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures, Version 2.0, Ada Joint Program Office, May 1989.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Compiler Validation Procedures.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.

Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation of legal Ada programs with certain language constructs which cannot be verified at compile time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be

customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Janus/ADA, Version 2.1.3

ACVC Version: 1.10

Certificate Number: 890919W1.10153

Host Computer:

Machine:	Unisys 386
Operating System:	SCO Xenix Version 2.3.2
Memory Size:	640 Kilobytes

Target Computer:

Machine:	Unisys 386
Operating System:	SCO Xenix Version 2.3.2
Memory Size:	640 Kilobytes

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

a. Capacities.

(1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)

(2) The compiler correctly processes tests containing loop statements nested to 17 levels. (See tests D55A03A..H (8 tests).)

(3) The compiler rejects tests containing block statements nested to 65 levels. (See test D56001B.)

(4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to six levels. (See tests D64005E..G (3 tests).)

b. Predefined types.

(1) This implementation supports the additional predefined types LONG_INTEGER and LONG_FLOAT in package STANDARD. (See tests B86001T..Z (7 tests).)

c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

(1) None of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)

(2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

(3) This implementation uses no extra bits for extra precision and uses no extra bits for extra range. (See test C35903A.)

(4) Sometimes NUMERIC_ERROR is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

(5) NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

(6) Underflow is gradual. (See tests C45524A..Z (26 tests).)

d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

(1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z (26 tests).)

(2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z (26 tests).)

(3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

e. Array types.

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

For this implementation:

(1) Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises no exception. (See test C36003A.)

(2) CONSTRAINT_ERROR is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components with each component being a null array. (See test C36202A.)

(3) NUMERIC_ERROR is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components with each component being a null array. (See test C36202B.)

(4) A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises STORAGE_ERROR when the array objects are declared. (See test C52103X.)

(5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

(6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

(7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

(8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

f. Discriminated types.

(1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

g. Aggregates.

(1) In the evaluation of a multi-dimensional aggregate, index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)

(2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

(3) CONSTRAINT_ERROR is raised before all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

h. Pragmas.

(1) The pragma INLINE is not supported for functions or procedures. (See tests LA3004A..B (2 tests), EA3004C..D (2 tests), and CA3004E..F (2 tests).)

i. Generics.

(1) Generic specifications and bodies can be compiled in separate compilations. (See tests CA1012A, CA2009C, CA2009F, BC3204C, and BC3205D.)

(2) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

j. Input and output.

(1) Modes IN_FILE and OUT_FILE are supported for SEQUENTIAL_IO. (See tests CE2102D..E (2 tests), CE2102N, and CE2102P.)

(2) Modes IN_FILE, OUT_FILE, and INOUT_FILE are supported for DIRECT_IO. (See tests CE2102F, CE2102I..J (2 tests), CE2102R, CE2102T, and CE2102V.)

(3) Modes IN_FILE and OUT_FILE are supported for text files. (See tests CE3102E and CE3102I..K (3 tests).)

(4) RESET and DELETE operations are supported for SEQUENTIAL_IO. (See tests CE2102G and CE2102X.)

(5) RESET and DELETE operations are supported for DIRECT_IO. (See tests CE2102K and CE2102Y.)

(6) RESET and DELETE operations are supported for text files. (See tests CE3102F..G (2 tests), CE3104C, CE3110A, and CE3114A.)

(7) Overwriting to a sequential file does not truncate the file. (See test CE2208B.)

(8) Temporary sequential files are given names and not deleted when closed. (See test CE2108A.)

(9) Temporary direct files are not given names. (See test CE2108C.)

(10) Temporary text files are not given names. (See test CE3112A.)

(11) More than one internal file can be associated with each external file for sequential files when reading only. (See tests CE2107A..E (5 tests), CE2102L, CE2110B, and CE2111D.)

(12) More than one internal file can be associated with each external file for direct files when reading only. (See tests CE2107F..H (3 tests), CE2110D, and CE2111H.)

(13) More than one internal file can be associated with each external file for text files when reading only. (See tests CE3111A..E (5 tests), CE3114B, and CE3115A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 44 tests had been withdrawn because of test errors. The AVF determined that 379 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 192 executable tests that use floating-point precision exceeding that supported by the implementation or that contain a line which exceed the maximum input line length allowed by this implementation. Modifications to the code, processing, or grading for 34 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	128	1131	1959	10	22	44	3294
Inapplicable	1	7	356	7	6	2	379
Withdrawn	1	2	35	0	6	0	44
TOTAL	130	1140	2350	17	34	46	3717

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	198	576	544	240	170	99	160	331	131	36	252	275	282	3294
Inappl	14	73	136	8	2	0	6	1	6	0	0	94	39	379
Wdrn	1	1	0	0	0	0	0	2	0	0	1	35	4	44
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

3.4 WITHDRAWN TESTS

The following 44 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	C97116A	BC3009B	CD2A62D
CD2A63A	CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B
CD2A66C	CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D
CD2A76A	CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G
CD2A84M	CD2A84N	CD2B15C	CD2D11B	CD5007B	CD50110
ED7004B	ED7005C	ED7005D	ED7006C	ED7006D	CD7105A
CD7203B	CD7204B	CD7205C	CD7205D	CE2107I	CE3111C
CE3301A	CE3411B				

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 379 tests were inapplicable for the reasons indicated:

a. The following 192 tests are not applicable because they have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

C24113L..P (5) C35705L..Y (14) C35706L..Y (14) C35707L..Y (14)
C35708L..Y (14) C35802L..Z (15) C45241L..Y (14) C45321L..Y (14)

C45421L..Y (14) C45521L..Z (15) C45524L..Z (15) C45621L..Z (15)
C45641L..Y (14) C46012L..Z (15)

b. C24113Q..Y (9 tests) are not applicable because they have contain a line which exceeds maximum input line length allowed by this implementation.

c. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT_FLOAT.

d. The following 30 tests are not applicable because this implementation does not support 'STORAGE_SIZE representation clauses for access types:

A39005C	C87B62B	CD1009J	CD1009R
CD1009S	CD1C03C	CD2A83A..C (3)	CD2A83E..F (2)
CD2A84B..I (8)	CD2A84K..L (2)	CD2B11B..G (6)	CD2B15B
CD2B16A	ED2A86A		

e. The following 16 tests are not applicable because this implementation does not support a predefined type SHORT_INTEGER:

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	B86001V
CD7101E				

f. C45231D, B86001X, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG_INTEGER, or SHORT_INTEGER.

g. C45531M..P (4 tests) and C45532M..P (4 tests) are not applicable because the value of SYSTEM.MAX_MANTISSA is less than 48.

h. D55A03E..H (4 tests) use 31 levels of loop nesting which exceeds the capacity of the compiler.

i. D56001B uses 65 levels of block nesting which exceeds the capacity of the compiler.

j. D64005F and D64005G are not applicable because this implementation does not support nesting 10 levels of recursive procedure calls.

k. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.

l. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG_FLOAT, or SHORT_FLOAT.

m. C96005B is not applicable because there are no values of type DURATION'BASE that are outside the range of DURATION.

n. LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are not applicable because this implementation does not support pragma INLINE.

o. CD1009C, CD2A41A..B (2 tests), CD2A41E, and CD2A42A..J (10 tests) are not applicable because this implementation does not support size clauses for floating point types.

p. The following 13 tests are not applicable because this implementation does not support record representation clauses:

CD1009N CD1009X..Z (3) CD1C03H CD1C04E CD4031A
CD4041A CD4051A..D (4) ED1D04A

q. The following 21 tests are not applicable because this implementation does not support size clauses for arrays:

CD2A61A..D (4) CD2A61F CD2A61H..L (5) CD2A62A..C (3)
CD2A64A..D (4) CD2A65A..D (4)

r. The following 16 tests are not applicable because this implementation does not support size clauses for records:

CD2A71A..D (4) CD2A72A..D (4) CD2A74A..D (4) CD2A75A..D (4)

s. CE2102D is inapplicable because this implementation supports CREATE with IN_FILE mode for SEQUENTIAL_IO.

t. CE2102E is inapplicable because this implementation supports CREATE with OUT_FILE mode for SEQUENTIAL_IO.

u. CE2102F is inapplicable because this implementation supports CREATE with INOUT_FILE mode for DIRECT_IO.

v. CE2102I is inapplicable because this implementation supports CREATE with IN_FILE mode for DIRECT_IO.

w. CE2102J is inapplicable because this implementation supports CREATE with OUT_FILE mode for DIRECT_IO.

x. CE2102N is inapplicable because this implementation supports OPEN with IN_FILE mode for SEQUENTIAL_IO.

y. CE2102O is inapplicable because this implementation supports RESET with IN_FILE mode for SEQUENTIAL_IO.

z. CE2102P is inapplicable because this implementation supports OPEN with OUT_FILE mode for SEQUENTIAL_IO.

aa. CE2102Q is inapplicable because this implementation supports RESET with OUT_FILE mode for SEQUENTIAL_IO.

ab. CE2102R is inapplicable because this implementation supports OPEN with INOUT_FILE mode for DIRECT_IO.

ac. CE2102S is inapplicable because this implementation supports RESET with INOUT_FILE mode for DIRECT_IO.

ad. CE2102T is inapplicable because this implementation supports OPEN with IN_FILE mode for DIRECT_IO.

ae. CE2102U is inapplicable because this implementation supports RESET with IN_FILE mode for DIRECT_IO.

af. CE2102V is inapplicable because this implementation supports OPEN with OUT_FILE mode for DIRECT_IO.

ag. CE2102W is inapplicable because this implementation supports RESET with OUT_FILE mode for DIRECT_IO.

ah. CE2107B..E (4 tests), CE2107L, CE2110B, and CE2111D are not applicable because multiple internal files cannot be associated with the same external file when one or more files is writing for sequential files. The proper exception is raised when multiple access is attempted.

ai. CE2107G..H (2 tests), CE2110D, and CE2111H are not applicable because multiple internal files cannot be associated with the same external file when one or more files is writing for direct files. The proper exception is raised when multiple access is attempted.

aj. EE2201D and EE2401D are not applicable because USE_ERROR is raised when trying to create a file with constrained array types.

ak. CE3102E is inapplicable because this implementation supports CREATE with IN_FILE mode for text files.

al. CE3102F is inapplicable because this implementation supports RESET for text files.

am. CE3102G is inapplicable because this implementation supports deletion of an external file for text files.

an. CE3102I is inapplicable because this implementation supports CREATE with OUT_FILE mode for text files.

ao. CE3102J is inapplicable because this implementation supports OPEN with IN_FILE mode for text files.

ap. CE3102K is inapplicable because this implementation supports OPEN with OUT_FILE mode for text files.

aq. CE3111B, CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal files cannot be associated with the same external file when one or more files is writing for text files. The proper exception is raised when multiple access is attempted.

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 34 tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B22003A	B24007A	B24009A	B29001A	B37106A	B49003A
B49005A	B51001A	B53003A	B55A01A	B63001A	B63001B
B91001H	BA1101A	BA1101C	BA1101E	BA3006A	BA3006B
BA3007B	BA3008A	BA3008B	BA3013A	BC2001D	BC2001E
BC3005B					

CC3601A was split because the original test generates more code than this implementation can handle in one compilation unit.

The following modifications were made to compensate for legitimate implementation behavior:

a. At the recommendation of the AVO, a "PRAGMA ELABORATE (REPORT);" was added at the beginning of C39005A to ensure that the elaboration of the routines in package REPORT takes place before these routines are called.

b. At the recommendation of the AVO, the expression "2**T'MANTISSA - 1" on line 262 in test CC1223A was changed to "(2**(T'MANTISSA - 1) - 1 + 2**(T'MANTISSA - 1))" in order to avoid generating the exception raising value 2**31. The grading criteria for this test were also modified. See the next section for modified evaluation criteria.

c. At the recommendation of the AVO, the variables V and W on line 41 of test CD2C11A were initialized to 5.0 due to PROGRAM_ERROR being raised when an attempt is made to use the uninitialized variables.

d. At the recommendation of the AVO, the lines which check whether temporary files can be created in tests CE2108B, CE2108D, and CE3112B were commented out because of the way in which temporary file names are constructed.

The following tests were graded using modified evaluation criteria:

a. At the recommendation of the AVO, test C34006D is graded PASSED provided the only failure messages arise from the requirements on the value of T'SIZE, where T is a type, since the meaning of 'SIZE applied to a type is not clear in this test.

b. At the recommendation of the AVO, test CC1223A is graded PASSED provided the only failure messages arise from the requirements on the value of T'AFT, where T is a type.

c. CE3804G writes, then reads, a floating-point literal and tests the input value against a textually identical literal; this implementation stores the numeric literal with greater precision than it uses for objects of the type, and because the literal is not a model number, the test for equality at line 121 fails. The AVO ruled that this test should be graded as passed if the only failure messages arise from the test at line 121.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the Janus/ADA, Version 2.1.3, compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the Janus/ADA, Version 2.1.3, compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	Unisys 386
Host operating system:	SCO Xenix, Version 2.3.2
Target computer:	Unisys 386

TEST INFORMATION

Target operating system:	SCO Xenix, Version 2.3.2
Compiler:	Janus/ADA, Version 2.1.3

Diskettes containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the diskettes. Tests requiring modifications during the prevalidation testing were included in their modified form on the diskettes.

The contents of the diskettes were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the Unisys 386. Results were printed from the host computer.

The compiler was tested using command scripts provided by R.R. Software, Inc. and reviewed by the validation team. The compiler was tested using all the following option settings. See Appendix E for a complete listing of the compiler options for this implementation. The following list of compiler options includes those options which were invoked by default:

-Q	Quiet error messages - suppresses user prompting on errors.
-W	Warnings off.
-T	Trimming code on - this directs the compiler to generate code which allows the linker to trim unused subprograms.
-D	Debugging code off.
-S?	Re-direct the compiler scratch files into a RAM disk where possible (? is replaced by a drive letter).
-E	Produce an .EXE file rather than a .COM file.
-F	Library calls are generated for floating point operations.
-L	No listing file is generated.
-O	Memory model 0 is used.
-R	The JRL file is put on the same disk as the input file.
-X	Extra symbol table information is not generated.
-Z	Optimization is done only where so specified by pragmas.

Tests were compiled, linked, and executed (as appropriate) using a single computer. Test output, compilation listings, and job logs were captured on diskettes and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Madison WI and was completed on 19 September 1989.

APPENDIX A

DECLARATION OF CONFORMANCE

R.R. Software, Inc. has submitted the following Declaration of Conformance concerning the Janus/ADA, Version 2.1.3, compiler.

DECLARATION OF CONFORMANCE

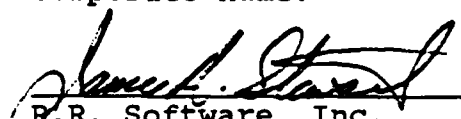
Compiler Implementor: R.R. Software, Inc.
Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB, OH 45433-6503
Ada Compiler Validation Capability (ACVC) Version: 1.10

Base Configuration

Base Compiler Name: Janus/ADA	Version: 2.1.3
Host Architecture ISA: Unisys 386	OS&VER #: SCO Xenix 2.3.2
Target Architecture ISA: Unisys 386	OS&VER #: SCO Xenix 2.3.2

Implementor's Declaration

I, the undersigned, representing R.R. Software, Inc., have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that R.R. Software, Inc. is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.



Date: September 12, 1989

R.R. Software, Inc.
James A. Stewart, Vice President

Owner's Declaration

I, the undersigned, representing R.R. Software, Inc., take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

Date: September 12, 1989

R.R. Software, Inc.
James A. Stewart, Vice President

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in Chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the Janus/ADA, Version 2.1.3, compiler, as described in this Appendix, are provided by R.R. Software, Inc. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

   ...

   type INTEGER is range -32768 .. 32767;
   type LONG_INTEGER is range -2147483648 .. 2147483647;

   type FLOAT is digits 6 range -((2.0 ** 128) - (2.0 ** 104)) ..
                                ((2.0 ** 128) - (2.0 ** 104));
   type LONG_FLOAT is digits 15 range -((2.0 ** 1024) - (2.0 ** 971)) ..
                                      ((2.0 ** 1024) - (2.0 ** 971));

   type DURATION is delta 0.00025 range -((2.0 ** 31) - 1)/4096.0 ..
                                         ((2.0 ** 31) - 1)/4096.0;

   ...

end STANDARD;
```

F Implementation Dependencies

This appendix specifies certain system-dependent characteristics of Janus/Ada, version 2.1.3 386 SCO XENIX compiler.

F.1 Implementation Dependent Pragmas

In addition to the required Ada pragmas, Janus/Ada also provides several others. Some of these pragmas have a *textual range*. Such pragmas set some value of importance to the compiler, usually a flag that may be On or Off. The value to be used by the compiler at a given point in a program depends on the parameter of the most recent relevant pragma in the text of the program. For flags, if the parameter is the identifier On, then the flag is on; if the parameter is the identifier Off, then the flag is off; if no such pragma has occurred, then a default value is used.

The range of a pragma - even a pragma that usually has a textual range - may vary if the pragma is not inside a compilation unit. This matters only if you put multiple compilation units in a file. The following rules apply:

1) If a pragma is inside a compilation unit, it affects only that unit.
2) If a pragma is outside a compilation unit, it affects all following compilation units in the compilation.

Certain required Ada pragmas, such as INLINE, would follow different rules; however, as it turns out, Janus/Ada ignores all pragmas that would follow different rules.

The following system-dependent pragmas are defined by Janus/Ada. Unless otherwise stated, they may occur anywhere that a pragma may occur.

ALL_CHECKS — Takes one of two identifiers On or Off as its argument, and has a textual range. If the argument is Off, then this pragma causes suppression of arithmetic checking (like pragma ARITHCHECK - see below), range checking (like pragma RANGECHECK - see below), storage error checking, and elaboration checking. If the argument is On, then these checks are all performed as usual. Note that pragma ALL_CHECKS does not affect the status of the DEBUG pragma; for the fastest run time code (and the worst run time checking), both ALL_CHECKS and DEBUG should be turned Off and the pragma OPTIMIZE (Time) should be used. Note also that ALL_CHECKS does not affect the status of the ENUMTAB pragma. Combining check

suppression using the pragma ALL_CHECKS and using the pragma SUPPRESS may cause unexpected results; it should not be done. However, ALL_CHECKS may be combined with the Janus/Ada pragmas ARITHCHECK and RANGECHECK; whichever relevant pragma has occurred most recently will determine whether a given check is performed. ALL_CHECKS is on by default. Turning any checks off may cause unpredictable results if execution would have caused the corresponding assumption to be violated. Checks should be off only in fully debugged and tested programs. After checks are turned off, full testing should again be done, since any program that handles an exception may expect results that will not occur if no checking is done.

ARITHCHECK — Takes one of the two identifiers On or Off as its argument, and has a textual range. Where ARITHCHECK is on, the compiler is permitted to (and generally does) not generate checks for situations where it is permitted to raise NUMERIC_ERROR; these checks include overflow checking and checking for division by zero. Combining check suppression using the pragma ARITHCHECK and using the pragma SUPPRESS may cause unexpected results; it should not be done. However, ARITHCHECK may be combined with the Janus/Ada pragma ALL_CHECKS; whichever pragma has occurred most recently will be effective. ARITHCHECK is on by default. Turning any checks off may cause unpredictable results if execution would have caused the corresponding assumption to be violated. Checks should be off only in fully debugged and tested programs. After checks are turned off, full testing should again be done, since any program that handles an exception may expect results that will not occur if no checking is done.

CLEANUP — Takes an integer literal in the range 0..3 as its argument, and has a textual range. Using this pragma allows the Janus/Ada run-time system to be less than meticulous about recovering temporary memory space it uses. This pragma can allow for smaller and faster code, but can be dangerous; certain constructs can cause memory to be used up very quickly. The smaller the parameter, the more danger is permitted. A value of 3 - the default value - causes the run-time system to be its usual immaculate self. A value of 0 causes no reclamation of temporary space. Values of 1 and 2 allow compromising between "cleanliness" and speed. Using values other than 3 adds some risk of your program running out of memory, especially in loops which contain certain constructs.

DEBUG

Takes one of the two identifiers On or Off as its argument, and has a textual range. This pragma controls the generation of line number code and procedure name code. When DEBUG is on, such code is generated. When DEBUG is off, no line number code or procedure names are generated. This information is used by the walkback which is generated after a run-time error (e.g., an unhandled exception). The walkback is still generated when DEBUG is off, but the line numbers will be incorrect, and no subprogram names will be printed. DEBUG's initial state can be set by the command line; if no explicit option is given, then DEBUG is initially on. Turning DEBUG off saves space, but causes the loss of much of Janus/Ada's power in describing run time errors.

Notes:
DEBUG should only be turned off when the program has no errors. The information provided on an error when DEBUG is off is not very useful.

If DEBUG is on at the beginning of a subprogram or package specification, then it must be on at the end of the specification. Conversely, if DEBUG is off at the beginning of such a specification, it must be off at the end. If you want DEBUG to be off for an entire compilation, then you can either put a DEBUG pragma in the context clause of the compilation or you can use the appropriate compiler option.

ENUMTAB

Takes one of the two identifiers On or Off as its argument, and has a textual range. This pragma controls the generation of enumeration tables. Enumeration tables are used for the attributes IMAGE, VALUE, and WIDTH, and hence to input and output enumeration values. The tables are generated when ENUMTAB is on. The state of the ENUMTAB flag is significant only at enumeration type definitions. If this pragma is used to prevent generation of a type's enumeration tables, then using the three mentioned attributes causes an erroneous program, with unpredictable results; furthermore, the type should not be used as a generic actual discrete type, and in particular TEXT_IO.ENUMERATION_IO should not be instantiated for the type. If the enumeration type is not needed for any of these purposes, the tables, which use a lot of space, are unnecessary. ENUMTAB is on by default.

PAGE_LENGTH — This pragma takes a single integer literal as its argument. It says that a page break should be added to the listing after each occurrence of the given number of lines. The default page length is 32000, so that no page breaks are generated for most programs. Each page starts with a header that looks like the following:

Janus/Ada Version 2.1.3 compiling *file* on *date* at *time*

RANGECHECK — Takes one of the two identifiers On or Off as its argument, and has a textual range. Where RANGECHECK is off, the compiler is permitted to (and generally does) not generate checks for situations where it is expected to raise CONSTRAINT_ERROR; these checks include null pointer checking, discriminant checking, index checking, array length checking, and range checking. Combining check suppression using the pragma RANGECHECK and using the pragma SUPPRESS may cause unexpected results; it should not be done. However, RANGECHECK may be combined with the Janus/Ada pragma ALL_CHECKS; whichever pragma has occurred most recently will be effective. RANGECHECK is on by default. Turning any checks off may cause unpredictable results if execution would have caused the corresponding assumption to be violated. Checks should be off only in fully debugged and tested programs. After checks are turned off, full testing should again be done, since any program that handles an exception may expect results that will not occur if no checking is done.

SYSLIB — This pragma tells the compiler that the current unit is one of the standard Janus/Ada system libraries. It takes as a parameter an integer literal in the range 1 .. 15; only the values 1 through 4 are currently used. For example, system library number 2 provides floating point support. Do not use this pragma unless you are writing a package to replace one of the standard Janus/Ada system libraries.

VERBOSE — Takes On or Off as its argument, and has a textual range. VERBOSE controls the amount of output on an error. If VERBOSE is on, the two lines preceding the error are printed, with an arrow pointing at the error. If VERBOSE is off, only the line number is printed.

VERBOSE(Off):

```
Line 16 at Position 5
*ERROR* Identifier is not defined
```

VERBOSE(On):

```
 15: if X = 10 then
 16:     Z := 10;
---------^
*ERROR* Identifier is not defined
```

The reason for this option is that an error message with VERBOSE on can take a long time to be generated, especially in a large program. VERBOSE's initial condition can be set by the compiler command line.

Several required Ada pragmas may have surprising effects in Janus/Ada. The PRIORITY pragma may only take the value 0, since that is the only value in the range System.Priority. Specifying any OPTIMIZE pragma turns on optimization; otherwise, optimization is only done if specified on the compiler's command line. The SUPPRESS pragma is ignored unless it only has one parameter. Also, the following pragmas are always ignored: CONTROLLED, INLINE, MEMORY_SIZE, PACK, SHARED, STORAGE_UNIT, and SYSTEM_NAME. Pragma CONTROLLED is always ignored because Janus/Ada does no automatic garbage collection; thus, the effect of pragma CONTROLLED already applies to all access types. Pragma SHARED is similarly ignored: Janus/Ada's non-preemptive task scheduling gives the appropriate effect to all variables. The pragmas INLINE, PACK, and SUPPRESS (with two parameters) all provide recommendations to the compiler; as Ada allows, the recommendations are ignored. The pragmas MEMORY_SIZE, STORAGE_UNIT, and SYSTEM_NAME all attempt to make changes to constants in the System package; in each case, Janus/Ada allows only one value, so that the pragma is ignored.

F.2 Implementation Dependent Attributes

Janus/Ada does not provide any attributes other than the required Ada attributes.

F.3 Specification of the Package SYSTEM

The package System for Janus/Ada has the following definition.

```
package System is

    -- System package for Janus/Ada

    -- Types to define type Address.
    type Offset_Type is new Long_Integer;
    type Word is range 0 .. 65536;
    for Word'Size use 16;
    type Address is record
        Offset : Offset_Type;
        Segment : Word;
    end record;
    Function "+" (Left : Address; Right : Offset_Type) Return Address;
    Function "+" (Left : Offset_Type; Right : Address) Return Address;
    Function "-" (Left : Address; Right : Offset_Type) Return Address;
    Function "-" (Left, Right : Address) Return Offset_Type;

    type Name is (UNIX);

    System_Name : constant Name := UNIX;

    Storage_Unit : constant := 8;
    Memory_Size : constant := 65536;
        -- Note: The actual memory size of a program is determined
        -- dynamically; this is the maximum number of bytes in the
data
        -- segment.

    -- System Dependent Named Numbers:
    Min_Int : constant := -2_147_483_648;
    Max_Int : constant :=  2_147_483_647;
    Max_Digits : constant := 15;
    Max_Mantissa : constant := 31;
    Fine_Delta : constant := 2#1.0#E-31;
        -- equivalently, 4.65661287307739257812 5E-10
    Tick : constant := 0.01; -- Some machines have less accuracy;
        -- for example, the IBM PC actually ticks about
        -- every 0.06 seconds.
```

```
    -- Other System Dependent Declarations
    subtype Priority is Integer range 0..0;

    type Byte is range 0 .. 255;
    for Byte'Size use 8;

end System;
```

The type Byte in the System package corresponds to the 8-bit machine byte. The type Word is a 16-bit Unsigned Integer type, corresponding to a machine word.

F.4 Restrictions on Representation Clauses

If T is a discrete type, or a fixed point type, then the size expression can give any value between 1 and 1000 bits (subject, of course, to allowing enough bits for every possible value). For other types, the expression must give the default size for T.

A length clause that specifies T'STORAGE_SIZE for an access type is not supported; Janus/Ada uses a single large common heap.

A length clause that specifies T'STORAGE_SIZE for a task type T is supported. Any integer value can be specified. Values smaller than 256 will be rounded up to 256 (the minimum T'Storage_Size), as the Ada standard does not allow raising an exception in this case.

A length clause that specifies T'SMALL for a fixed point type must give a value (subject to the Ada restrictions) in the range

```
        2.0 ** (-99) .. 2.0 ** 99,
```

inclusive.

An enumeration representation clause for a type T may give any integer values within the range System.Min_Int .. System.Max_Int. If a size length clause is not given for the type, the type's size is determined from the literals given. (If all of the literals fit in a byte, then Byte'Size is used; similarly for Integer and Long_Integer).

The expression in an alignment clause in a record representation clause must equal 1.

- A component clause must give a storage place that is equivalent to the default value of the POSITION attribute for such a component.

  A component clause must give a range that starts at zero and extends to one less than the size of the component.

  Janus/Ada supports address clauses on most objects. Address clauses are not allowed on parameters, generic formal parameters, and renamed objects. The address given for an object address clause may be any legal value of type System.Address. It will be interpreted as an absolute machine address, using the segment part as a selector if in the protected mode. It is the user's responsibility to ensure that the value given makes sense (i.e., points at memory, does not overlay other objects, etc.) No other address clauses are supported.

F.5 Implementation Defined Names

Janus/Ada uses no implementation generated names.

F.6 Address Clause Expressions

The address given for an object address clause may be any legal value of type System.Address. It will be interpreted as an absolute machine address, using the segment part as a selector if in the protected mode. It is the user's responsibility to ensure that the value given makes sense (i.e., points at memory, does not overlay other objects, etc.)

F.7 Unchecked_Conversion Restrictions

We first make the following definitions:
A type or subtype is said to be a *simple type* or a *simple subtype* (respectively) if it is a scalar (sub)type, an access (sub)type, a task (sub)type, or if it satisfies the following two conditions:

1) If it is an array type or subtype, then it is constrained and its index constraint is static; and
2) If it is a composite type or subtype, then all of its subcomponents have a simple subtype.

A (sub)type which does not meet these conditions is called *non-simple*. Discriminated records can be simple; variant records can be simple. However, constraints which depend on discriminants are non-simple (because they are non-static).

Janus/Ada imposes the following restriction on instantiations of Unchecked_Conversion: for such an instantiation to be legal, both the source actual subtype and the target actual subtype must be simple subtypes, and they must have the same size.

F.8 Implementation Dependencies of I/O

The syntax of an external file name depends on the operating system being used. Some external files do not really specify disk files; these are called *devices*. Devices are specified by special file names, and are treated specially by some of the I/O routines.

The syntax of an XENIX filename is:

```
[path]filename
```

where "path" is an optional path consisting of directory names, each followed by a foreslash; "filename" is the filename (maximum 14 characters). See your XENIX manual for a complete description. In addition, the following special device names are recognized:

/dev/sti	XENIX standard input. The same as Standard_Input. Input is buffered by lines, and all XENIX line editing characters may be used. Can only be read.
/dev/sto	XENIX standard output. The same as Standard_Output. Can only be written.
/dev/err	XENIX standard error. The output to this device cannot be redirected. Can only be written.
/dev/ekbd	The current terminal input device. Single character input with echoing. Due to the design of XENIX, this device *can* be redirected. Can be read and written.
/dev/kbd	The current terminal input device. No character interpretation is performed, and there is no character echo. Again, the input to this device can be redirected, so it does not *always* refer to the physical keyboard.

The XENIX device files may also be used.

The XENIX I/O system will do a search of the default search path (set by the environment PATH variable) if the following conditions are met:

1) No path is present in the file name; and
2) The name is not that of a device.

Alternatively, you may think of the search being done if the file name does not contain any of the characters ':' or '/'.

The default search path cannot be changed while the program is running, as the path is copied by the Janus/Ada program when it starts running.

Note:
Creates will never cause a path search as they must work in the current directory.

Upon normal completion of a program, any open external files are closed. Nevertheless, to provide portability, we recommend explicitly closing any files that are used.

Sharing external files between multiple file objects causes the corresponding external file to be opened multiple times by the operating system. The effects of this are defined by your operating system. This external file sharing is only allowed if all internal files associated with a single external file are opened only for reading (mode In_File), and no internal file is Created. Use_Error is raised if these requirements are violated. A Reset to a writing mode of a file already opened for reading also raise Use_Error if the external file also is shared by another internal file.

Binary I/O of values of access types will give meaningless results and should not be done. Binary I/O of types which are not simple types (see definition in Section F.7, above) will raise Use_Error when the file is opened. Such types require specification of the block size in the form, a capability which is not yet supported.

The form parameter for Sequential_IO and Direct_IO is always expected to be the null string.

The type Count in the generic package Direct_IO is defined to have the range 0 .. 2_147_483_647.

Ada specifies the existence of special markers called *terminators* in a text file. Janus/Ada defines the line terminator to be <LF> (line feed), with or without an additional <CR> (carriage return). The page terminator is the <FF> (form feed) character; if it is not preceded by a <LF>, a line terminator is also assumed.

The file terminator is the end-of-file returned by the host operating system. If no line and/or page terminator directly precedes the file terminator, they are assumed. The only legal form for text files is "" (the null string). All other forms raise USE_ERROR.

Output of control characters does not affect the layout that Text_IO generates. In particular, output of a <LF> before a New_Page does not suppress the New_Line caused by the New_Page.

The character <LF> is written to represent the line terminator.

The type Text_IO.Count has the range 0 .. 32767; the type Text_IO.Field also has the range 0 .. 32767.

IO_Exceptions.USE_ERROR is raised if something cannot be done because of the external file system; such situations arise when one attempts:

- to create or open an external file for writing when the external file is already open (via a different internal file).
- to create or open an external file when the external file is already open for writing (via a different internal file).
- to reset a file to a writing mode when the external file is already open (via a different internal file).
- to write to a full device (Write, Close);
- to create a file in a full directory (Create);
- to have more files open than the OS allows (Open, Create);
- to open a device with an illegal mode;
- to create, reset, or delete a device;
- to create a file where a protected file (i.e., a directory or read-only file) already exists;
- to delete a protected file;
- to use an illegal form (Open, Create); or
- to open a file for a non-simple type without specifying the block size;
- to open a device for direct I/O.

IO_Exceptions.DEVICE_ERROR is raised if a hardware error other than those covered by USE_ERROR occurs. These situations should never occur, but may on rare occasions. For example, DEVICE_ERROR is raised when:

- a file is not found in a Close or a Delete;
- a seek error occurs on a direct Read or Write; or
- a seek error occurs on a sequential End_Of_File.

The subtypes Standard.Positive and Standard.Natural, used by some I/O routines, have the maximum value 32767.

No package Low_Level_IO is provided.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
$BIG_ID1 An identifier the size of the maximum input line length which is identical to $BIG_ID2 except for the last character.	(1..199 => 'A', 200 => '1')
$BIG_ID2 An identifier the size of the maximum input line length which is identical to $BIG_ID1 except for the last character.	(1..199 => 'A', 200 => '2')
$BIG_ID3 An identifier the size of the maximum input line length which is identical to $BIG_ID4 except for a character near the middle.	(1..99 => 'A', 100 => '3', 101..200 => 'A')

TEST PARAMETERS

Name and Meaning	Value
$BIG_ID4 An identifier the size of the maximum input line length which is identical to $BIG_ID3 except for a character near the middle.	(1..99 => 'A', 100 => '4', 101..200 => 'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197 => '0', 198..200 => "298")
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..195 => '0', 196..200 => "690.0")
$BIG_STRING1 A string literal which when catenated with $BIG_STRING2 yields the image of $BIG_ID1.	(1 => '"', 2..101 => 'A', 102 => '"')
$BIG_STRING2 A string literal which when catenated to the end of $BIG_STRING1 yields the image of $BIG_ID1.	(1 => '"', 2..100 => 'A', 101 => '1', 102 => '"')
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..180 => ' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	32767
$DEFAULT_MEM_SIZE An integer literal whose value is SYSTEM.MEMORY_SIZE.	65536
$DEFAULT_STOR_UNIT An integer literal whose value is SYSTEM.STORAGE_UNIT.	8

Name and Meaning	Value
$DEFAULT_SYS_NAME The value of the constant SYSTEM.SYSTEM_NAME.	UNIX
$DELTA_DOC A real literal whose value is SYSTEM.FINE_DELTA.	0.000000000465661287307739257812 5
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	32767
$FIXED_NAME The name of a predefined fixed-point type other than DURATION.	NOT_APPLICABLE
$FLOAT_NAME The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	NOT_APPLICABLE
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	300000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	1.0E6
$HIGH_PRIORITY An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	0
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	/NODIRECTORY/FILENAME
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	<BAD|^^>
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768

TEST PARAMETERS

Name and Meaning	Value
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-305000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-1.0E6
$LOW_PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	0
$MANTISSA_DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	200
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be $MAX_IN_LEN long.	(1..2 => "2:", 3..197 => '0', 198..200 => "11:")

Name and Meaning	Value
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be $MAX_IN_LEN long.	(1..3 => "16:", 4..196 => '0', 197..200 => "F.E:")
$MAX_STRING_LITERAL A string literal of size $MAX_IN_LEN, including the quote characters.	(1 => '"', 2..201 => 'A', 202 => '"')
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$MIN_TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.	32
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	NOT_APPLICABLE
$NAME_LIST A list of enumeration literals in the type SYSTEM.NAME, separated by commas.	UNIX
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFF#
$NEW_MEM_SIZE An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than $DEFAULT_MEM_SIZE. If there is no other value, then use $DEFAULT_MEM_SIZE.	65536

TEST PARAMETERS

Name and Meaning	Value
$NEW_STOR_UNIT An integer literal whose value is a permitted argument for pragma STORAGE_UNIT, other than $DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
$NEW_SYS_NAME A value of the type SYSTEM.NAME, other than $DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	UNIX
$TASK_SIZE An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	32
$TICK A real literal whose value is SYSTEM.TICK.	0.01

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 44 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

a. E28005C: This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this text that must appear at the top of the page.

b. A39005G: This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).

c. B97102E: This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).

d. C97116A: This test contains race conditions, and it assumes that guards are evaluated indivisibly. A conforming implementation may use interleaved execution in such a way that the evaluation of the guards at lines 50 & 54 and the execution of task CHANGING_OF_THE_GUARD results in a call to REPORT.FAILED at one of lines 52 or 56.

e. BC3009B: This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).

f. CD2A62D: This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).

g. CD2A63A..D, CD2A66A..D, CD2A73A..D, and CD2A76A..D (16 tests): These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

h. CD2A81G, CD2A83G, CD2A84M..N, and CD50110 (5 tests): These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86, 96, and 58, respectively).

i. CD2B15C and CD7205C: These tests expect that a 'STORAGE_SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.

j. CD2D11B: This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.

k. CD5007B: This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).

l. ED7004B, ED7005C..D, and ED7006C..D (5 tests): These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.

m. CD7105A: This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).

n. CD7203B and CD7204B: These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

o. CD7205D: This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.

p. CE2107I: This test requires that objects of two similar scalar types be distinguished when read from a file--DATA_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid (line 90).

q. CE3111C: This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.

r. CE3301A: This test contains several calls to END_OF_LINE and END_OF_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD_INPUT (lines 103, 107, 118, 132, and 136).

s. CE3411B: This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.

APPENDIX E

COMPILER OPTIONS AS SUPPLIED BY R.R. SOFTWARE, INC.

Compiler: Janus/ADA, Version 2.1.3

ACVC Version: 1.10

F.9 Running the compiler and linker

The Janus/Ada compiler is invoked using the following format:

```
JANUS filename [-option]
```

where filename is an XENIX file name with optional compiler options [-option].

The compiler options are:

B Brief error messages. The line in error is not printed (equivalent to turning off pragma VERBOSE).

D Don't generate debugging code (equivalent to turning off pragma DEBUG)

F Use in-line 80387 instructions for Floating point operations. By default the compiler generates library calls for floating point operations. The 80387 may be used to execute the library calls. A floating point support library is still required, even though this option is used.

L Create a listing file with name filename.PRN on the same disk as filename. The listing file will be a listing of only the last compilation unit in a file.

Ox Object code memory model. X is 0 for the 80386 system. Other memory models are not supported. (Since this model 'limits' a program to 4 Gigabytes of Code and 4 Gigabytes of Data, this is not a concern). Memory model 0 is assumed if this option is not given.

Q Quiet error messages. This option causes the compiler not to wait for the user to interact after an error. In the usual mode, the compiler will prompt the user after each error to ask if the compilation should be aborted. This option is useful if the user wants to take a coffee break while the compiler is working, since all user prompts are suppressed. The errors (if any) will not stay on the screen when this option is used; therefore, the console traffic should be sent to the printer or to a file. Be warned that certain syntax errors can cause the compiler to print many error messages for each and every line in the program.

T Generate information which allows trimming unused subprograms from the code. This option tells the compiler to generate information which can be used by the remove subprograms from the final code. This option increases the size of the .JRL files produced. We recommend that it be used on reusable libraries of code (like trig. libraries or stack packages) - that is those compilations for which it is likely that some subprograms are not called.

W Don't print any warning messages. For more control of warning messages, use the following option form (Wx).

Wx Print only warnings of level less than the specified digit 'x'. The given value of x may be from 1 to 9. The more warnings you are willing to see, the higher the number you should give.

X Handle eXtra symbol table information. This is for the use of debuggers and other future tools. This option requires large quantities of memory and disk space, and thus should be avoided if possible.

Z Turn on optimization. This has the same effect as if the pragma OPTIMIZE were set to SPACE throughout your compilation.

The default values for the command line options are:

B Error messages are verbose.
D Debug code is generated.
F Library calls are generated for floating point operations.
L No listing file is generated.
O Memory model 0 is used.
Q The compiler prompts for abort after every error.
T No trimming code is produced.
W All warnings are printed.
X Extra symbol table information is not generated.
Z Optimization is done only where so specified by pragmas.

Leading spaces are disregarded between the filename and the call to JANUS. Spaces are otherwise not recommended on the command line. The presence of blanks to separate the options will be ignored.

Examples:

```
JANUS test-Q-L
JANUS test.run-W4
JANUS test
JANUS test   .run -B -W-L
```

The compiler produces a SYM (SYMbol table information) file when a specification is compiled, and a SRL or JRL (Specification ReLocatable or Janus ReLocatable) file when a body is compiled. To make an executable program, the appropriate SRL and JRL files must be *linked* (combined) with the run-time libraries. This is accomplished by running the Janus/Ada linker, JLINK.

The Janus/Ada linker is invoked using the following format:

```
JLINK filename [-option]
```

Here "filename" is the name of the SRL or JRL file created when the main program was compiled (without the .SRL or .JRL extension) with optional linker options [-option]. The filename usually corresponds to the first fourteen letters of the name of your main program. See the linker manual for more detailed directions. We summarize here, however, a few of the most commonly used linking options:

E Create an EXE file. This option has no effect on the 80386 linker (it always creates an EXE file).

F0 Use software floating point (the default).

F2 *Use hardware (80387) floating point.*

L Display lots of information about the loading process.

O0 Use memory model 0 (the default); see the description of the /O option in the compiler, above.

Q Use quiet error messages; i.e., don't wait for the user to interact after an error.

T Trim unused subprograms from the code. This option tells the linker to remove subprograms which are never called from the final output file. This option reduces space usage of the final file by as much as 30K.

Examples:

```
JLINK test
JLINK test -Q-L
JLINK test-L-F2
```

Note that if you do not have a hardware floating point chip, then you generally will not need to use any linker options.